

# BLUE SPIKE

## Ex. F. Infringement Chart for Claim 1 of the '276 Patent

---

SPOTIFY

BLUE SPIKE

## PATENT DETAILS

**U.S. Patent No.  
8,265,276**

**Method for combining transfer functions  
and predetermined key creation**

**ISSUE DATE :**

September 11, 2012

**FILING DATE :**

December 22, 2009

**INVENTOR :**

Scott A. Moskowitz;

**ABSTRACT:**

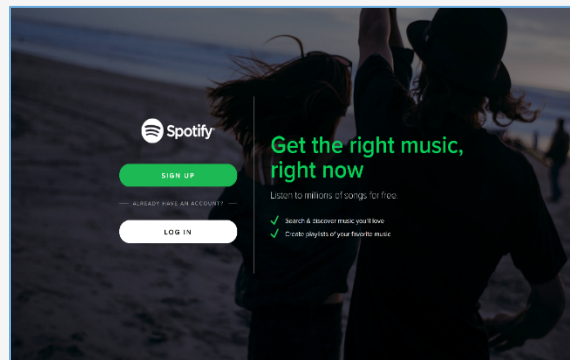
A method for combining transfer functions with predetermined key creation. In one embodiment, digital information, including a digital sample and format information, is protected by identifying and encoding a portion of the format information. Encoded digital information, including the digital sample and the encoded format information, is generated to protect the original digital information. In another embodiment, a digital signal, including digital samples in a file format having an inherent granularity, is protected by creating a predetermined key. The predetermined key is comprised of a transfer function-based mask set to manipulate data at the inherent granularity of the file format of the underlying digitized samples.

**CLAIM 1**

1. A method for protecting a digital signal, comprising the steps of:

**SPOTIFY**

*Spotify offers a method for protecting a digital signal (music content) through their premium media streaming services which can be played on any device. More specifically, Spotify uses a digital rights management system to encrypt the music content which is playable through the Spotify app only. For instance, to play the encrypted music on web browser, Spotify utilizes Encrypted Media Extensions. Spotify uses Google Widevine CDM to enable instant streaming on consumer devices, such as Chrome Browser on Desktop, Firefox Browser on Desktop, Chromecast, Android TVs etc.*



See Exhibit 1, p. 1, Spotify Web page (Date Accessed 04/27/2018), available at <https://open.spotify.com/browse>

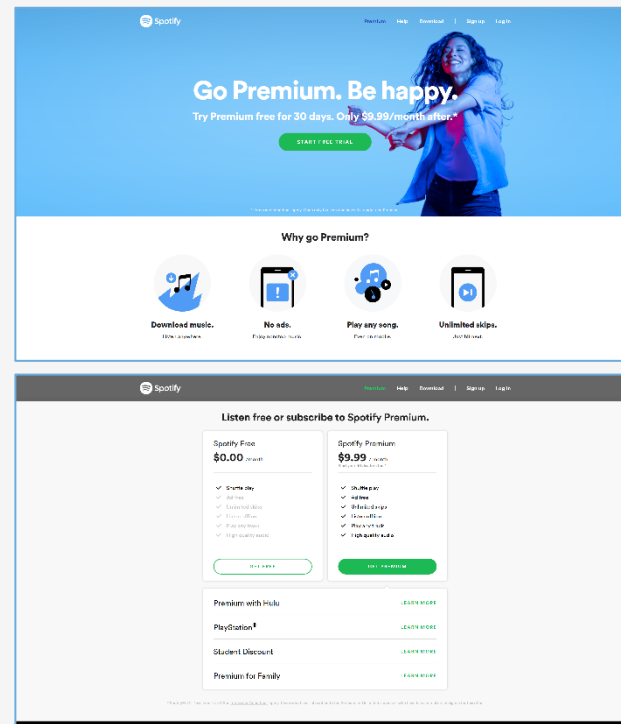
See Exhibit 2, "Discover and stream music across all your devices with Spotify \_ FileHippo News" webpage (Date Accessed 04/27/2018), available at <https://news.filehippo.com/2016/12/discover-and-stream-music-across-all-your-devices-with-spotify/>

## CLAIM 1

1. A method for protecting a digital signal, comprising the steps of:

## SPOTIFY

*Spotify offers Digital Rights Management to offer protection against illegal songs downloads through Encrypted Media Extensions (EME). It offers a freemium service to conduct trusted transactions between the user and the Spotify server. The subscription to Spotify allows playback of songs only through Web player or the application and cannot be accessed beyond them.*



See Exhibit 3, p. 1 & 2, "Spotify Premium" webpage (Date Accessed 04/30/2018), available at <https://www.spotify.com/us/premium/>

## CLAIM 1

1. A method for protecting a digital signal, comprising the steps of:

## SPOTIFY

## Create your account

No queues, no entry fees, and no wristbands. It couldn't be easier to get in and join Spotify.

Creating an account with Spotify gives you access to our [Free](#) service. For more great features, [try Premium](#).

**Tip:** To keep all your music and subscriptions in one place, we recommend not creating more than one account. Check out [how to find any accounts you already have](#).

## Sign up with your email address

1. Head to [Spotify.com/signup](#).
2. Enter your **Email**.
3. Confirm your email by entering it again in the next field.
4. Choose a **Password**.
5. Enter a name in **What should we call you?** if you'd like to personalize your account with a profile name.  
**Note:** It's not possible to log in with your profile name.
6. Complete the other details.
7. Click **SIGN UP**.

See Exhibit 4, p. 1, "Create your account - Spotify" webpage (04/30/2018), available at [https://support.spotify.com/us/account\\_payment\\_help/account\\_basics/create-your-spotify-account/](https://support.spotify.com/us/account_payment_help/account_basics/create-your-spotify-account/)

**"Spotify also plays through several sound systems, TVs, and car stereo systems.**

Please contact your device's manufacturer to determine whether it supports Spotify and, if so, how to enable it.

Note: **The Spotify app can vary across different devices and operating systems.** We recommend running the latest firmware on your device for the best experience."

See Exhibit 5, p. 1, "System requirements - Spotify" webpage (04/30/2018), available at <https://support.spotify.com/dk/article/spotify-system-requirements/> (emphasis added)

**CLAIM 1**

1. A method for protecting a digital signal, comprising the steps of:

**SPOTIFY**

## System requirements

The following are the system requirements for using Spotify and accessing Spotify content through the [Spotify app](#):

	Model	OS
<b>iPhone</b>	iPhone 4S or above	iOS 9 or above. 100 MB free space
<b>iPad</b>	iPad 2 or above	iOS 9 or above. 100 MB free space
<b>iPod</b>	5th generation iPod Touch or above	iOS 9 or above. 100 MB free space
<b>Android</b>	Any device	Android OS 4.1 or above. 500 MB free space
<b>Mac</b>	Any device	OS X 10.9 or above
<b>Windows</b>	Any device	Windows 7 or above

### “The web player

The web player is supported by the following web browsers:

**Chrome**  
**Firefox**  
**Edge**  
**Opera**

**Spotify content may only be accessed with the Spotify app, the web player website, or apps otherwise authorized by Spotify.”**

See Exhibit 5, p. 1, “System requirements - Spotify” webpage (04/30/2018), available at <https://support.spotify.com/dk/article/spotify-system-requirements/> (emphasis added)

**CLAIM 1**

1. A method for protecting a digital signal, comprising the steps of:

**SPOTIFY**

*Upon information and belief, services like Spotify uses DRM for playback of the content.*

“We **consume movies and music through services like Netflix, Spotify, Pandora, Apple Music, etc., often over the web. Netflix, et al. are obligated, due to agreements with content partners, to serve their content with DRM.** Moreover, web streaming is likely a significant chunk of usage for Netflix. This makes **the problem of enforcing DRM over the web an incredibly important problem for Netflix, et al. to solve.**

That’s where **EMEs come into play.** They are essentially **a technology that allows the browser to communicate with DRM systems over an encrypted medium.** They **make it possible for DRM software to work over the browser and relay encrypted content to the user.** This means **Netflix can serve you a movie that’s encrypted and protected by DRM.”**

See Exhibit 6, p. 2, “The most controversial HTML5 extension – LogRocket” blog post (Date Accessed 05/01/2018), available at <https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291> (emphasis added)

**CLAIM 1**

1. A method for protecting a digital signal, comprising the steps of:

**SPOTIFY**

*Upon information and belief, like Felix, the DRM implemented in Spotify works similarly.*

**“Say Felix is a movie streaming service that serves up movies over the browser and uses EMEs to protect the content it delivers.** Suppose that user wants to play some encrypted content from Felix. Here’s what happens:

**The browser will load up this content, realize it is encrypted and then trigger some Javascript code.**

**This Javascript will take the encrypted content and pass it along to something called the Content Decryption Module (CDM)—more on this later.**

**The CDM will then trigger a request that will be relayed by Felix to the license server.** This is a Felix-controlled server that determines whether or not a particular user is allowed to play a particular piece of content. **If the license server thinks our user can watch the content, it will send back a decryption key for the content to the CDM.**

**The CDM will then decrypt the content and it will begin playing for the user.”**

The most important **chunk of this whole process is the CDM.** By its nature, the **CDM has to be a piece of code that’s trusted by Felix.** If it isn’t, Felix can’t be sure that it will do what it wants it to. This means that the **CDM has to come with the browser.** There are a couple of different providers for CDMs, including **Google’s Widevine** and Adobe Primetime. Both of these provide the **encryption support necessary for content providers such as Felix to enforce DRM.**

See Exhibit 6, p. 2 & 3, “The most controversial HTML5 extension – LogRocket” blog post (Date Accessed 05/01/2018), available at <https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291> (emphasis added)



## CLAIM 1

1. A method for protecting a digital signal, comprising the steps of:

## SPOTIFY

“Web Playback SDK Quick Start BETA

Create a **simple web app with the Web Playback SDK that allows you to play an audio track inside your browser.**

By using Spotify developer tools, you accept our Developer Terms of Service. They contain important information about what you can and can't do with our developer tools. Please read them carefully.

Introduction

**The Web Playback SDK is client-side JavaScript library which allows you to create a new player in Spotify Connect and play any audio track from Spotify in the browser via Encrypted Media Extensions.**

You can read more about the SDK in the overview, or dig into the reference documentation. In this Quick Start, we will be adding the Web Playback SDK to a simple HTML page.

Authenticating with Spotify

**You will need an access token from your personal Spotify Premium account. Click the button below to quickly obtain your access token.”**

See Exhibit 7, p. 1, “Web Playback SDK Quick Start \_ Spotify for Developers” webpage (Date Accessed 05/01/2018), available at <https://beta.developer.spotify.com/documentation/web-playback-sdk/quick-start/> (emphasis added)

## CLAIM 1

1. A method for protecting a digital signal, comprising the steps of:

## SPOTIFY

**“Encrypted Media Extensions provides an API that enables web applications to interact with content protection systems, to allow playback of encrypted audio and video.**

EME is designed to **enable the same app and encrypted files to be used in any browser, regardless of the underlying protection system.** The former is made possible by the standardized APIs and flow while the latter is made possible by the concept of Common Encryption

EME is an extension to the HTMLMediaElement specification — hence the name. Being an 'extension' means that browser support for EME is optional: if a browser does not support encrypted media, it will not be able to play encrypted media, but EME is not required for HTML spec compliance

“EME implementations use the following external components:

**Key System: A content protection (DRM) mechanism. EME doesn't define Key Systems themselves, apart from Clear Key (more about that below).**

**Content Decryption Module (CDM): A client-side software or hardware mechanism that enables playback of encrypted media. As with Key Systems, EME doesn't define any CDMs, but provides an interface for applications to interact with CDMs that are available.**

**License (Key) server: Interacts with a CDM to provide keys to decrypt media. Negotiation with the license server is the responsibility of the application.**

*continued on next slide.*

## CLAIM 1

1. A method for protecting a digital signal, comprising the steps of:

## SPOTIFY

**Packaging service: Encodes and encrypts media for distribution/consumption.**

Note that an application **using EME interacts with a license server to get keys to enable decryption**, but user identity and authentication are not part of EME. **Retrieval of keys to enable media playback happens after (optionally) authenticating a user.** **Services such as Netflix must authenticate users within their web application: when a user signs into the application, the application determines the user's identity and privileges."**

See Exhibit 8, p. 1, "What is EME\_ \_ Web Fundamentals \_ Google Developers" forum page (Date Accessed 05/02/2018), available at <https://developers.google.com/web/fundamentals/media/eme> (emphasis added)

*Thus, this claim element is infringed literally, or in the alternative under the doctrine of equivalents.*

## CLAIM 1

providing a digital signal comprising digital data and file format information defining how the digital signal is encoded;

## SPOTIFY

*Spotify provides digital content (online music streaming) through its web player and App. The music is available in Ogg Vorbis format (digital signal encoded). Upon information and belief, Spotify makes available 96 Kbps – 160 Kbps Music bitrate for Spotify Free customers (file format information) and up to 320 Kbps for Spotify Premium customers.*

“Generally ou can only use Spotify to play offline music synced from Spotify.

**The files are Ogg Vorbis format but encrtyped by Spotify, so no other media scanner will recognize them as music files.”**

See Exhibit 25, p. 2, “Solved\_ Where does Spotify save the offline playlist files... - The Spotify Community” webpage (Date Accessed 05/07/2018), available at <https://community.spotify.com/t5/Android/Where-does-Spotify-save-the-offline-playlist-files/td-p/6929> (emphasis added)

**“With Spotify, it’s easy to find the right music for every moment – on your phone, your computer, your tablet and more.**

**There are millions of tracks on Spotify. So whether you’re working out, partying or relaxing, the right music is always at your fingertips. Choose what you want to listen to, or let Spotify surprise you.**

You can also browse through the music collections of friends, artists and celebrities, or create a radio station and just sit back.

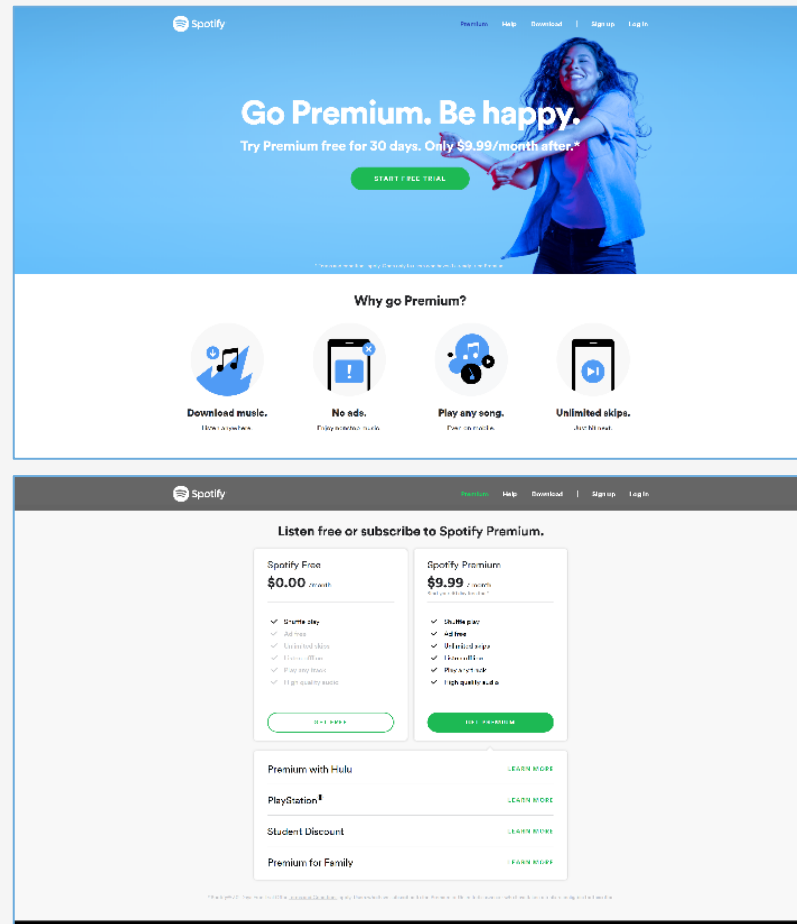
Soundtrack your life with Spotify. **Subscribe or listen for free.”**

See Exhibit 26, p. 1, “Contact - Spotify” webpage (Date Accessed 05/07/2018), available at <https://www.spotify.com/us/about-us/contact/> (emphasis added)

## CLAIM 1

providing a digital signal  
comprising digital data  
and file format information  
defining how the digital  
signal is encoded;

## SPOTIFY



See Exhibit 3, p. 1 & 2, "Spotify Premium" webpage (Date Accessed 04/30/2018), available at <https://www.spotify.com/us/premium/>

## CLAIM 1

providing a digital signal comprising digital data and file format information defining how the digital signal is encoded;

## SPOTIFY

### **“6. Audio Quality: 160Kbps VS 320Kbps**

On desktop, **default streaming audio quality for Spotify Free users is Ogg Vorbis 160kbit/s while Premium subscribers can choose to switch on High quality streaming, which uses 320kbit/s.** On iPhone, iPad or Android, Spotify Free users can choose either 96kbit/s or 160kbit/s while Premium subscribers have one more option which is 320kbit/s. As to streaming audio quality on Chromecast, it's AAC 128kbit/s for Spotify Free, and 256kbit/s for Spotify Premium.”

See Exhibit 27, p. 5, “Spotify Free VS Premium\_ 6 Differences You Need to Know” webpage (Date Accessed 05/08/2018), available at <http://www.tunemobie.com/resources/spotify-free-vs-premium-6-differences.html> (emphasis added)

### **“AUDIO QUALITY**

**Spotify uses three different quality settings for streaming, all in the Ogg Vorbis format. 96kbps is the standard bitrate for mobile, which then jumps to 160kbps for desktop and web player 'standard quality' and 'high quality' on mobile.**

**If you pay the monthly fee for Spotify Premium, you'll get 320kbps which is 'high quality' on desktop and 'extreme quality' on mobile.”**

See Exhibit 28, p. 3, “Spotify Free vs Spotify Premium - Tech Advisor” webpage (Date Accessed 05/08/2018), available at <https://www.techadvisor.co.uk/review/audio-and-music-software/spotify-free-vs-spotify-premium-3597766/> (emphasis added)

## CLAIM 1




providing a digital signal  
comprising digital data  
and file format information  
defining how the digital  
signal is encoded;

## SPOTIFY

### Desktop

The desktop app's standard quality is Ogg Vorbis 160kbit/s.




Premium subscribers can choose to switch on **High quality streaming**, which uses 320kbit/s:

1. Click the arrow  in the top-right corner and select **Settings**.
2. Under **Music Quality**, switch **High quality streaming (Premium only)** on , or off .

**Tip:** You can also ensure the same volume level is set for all songs in **Settings**. Click **SHOW ADVANCED SETTINGS** and find the option under **Playback**.

### iPhone and iPad

#### Audio Quality

1. Tap **Home** .  
Got Premium? Tap **Your Library** .
2. Tap **Settings** .
3. Tap **Music Quality**.
4. Select your preferred settings.

You can choose from the following audio quality settings, all in the Ogg Vorbis format:

- **Normal** – Equivalent to approximately 96kbit/s.
- **High** – Equivalent to approximately 160kbit/s.
- **Extreme** – Equivalent to approximately 320kbit/s.
- **Automatic** – Dependent on your network connection.

You can have different settings for listening online (Stream quality) or offline (Download quality). The higher the Stream quality, the more data will be used. The higher the Download quality, the more disk space used.

See Exhibit 29, p. 1, "Audio settings - Spotify" webpage (Date Accessed 05/08/2018), available at <https://support.spotify.com/is/article/high-quality-streaming/?ref=related>

## CLAIM 1

providing a digital signal comprising digital data and file format information defining how the digital signal is encoded;

## SPOTIFY

“In accordance with some implementations, a **method of providing media content** is disclosed herein. **The method is performed at a computer system** (e.g., a server system) with one or more processors and memory. **The method includes obtaining content-access information that enables distribution of content to a plurality of clients having different file format processing capabilities.** The method also includes providing to a first client, having first file format processing capabilities, first information that enables the first client to access respective content in a first file format.”

See Exhibit 30, at Col. 2:49-58, “U.S. Patent No. 9,529,888 - System and method for efficiently providing media and associated metadata.” Patent assigned to Spotify (Date Accessed 05/10/2018), available at <https://patentimages.storage.googleapis.com/88/d4/11/6b1fc7c1e09e6b/US9529888.pdf> (emphasis added)

*Thus, this claim element is infringed literally, or in the alternative under the doctrine of equivalents.*



## CLAIM 1

creating a predetermined key to manipulate the digital signal;

## SPOTIFY

*The digital content is manipulated using a predetermined key (here **Encrypted Media Extensions (EME)**) before transmission from Party 1 to Party 2 so that it is not compromised or deciphered by an unauthorized user. The package's encrypted content will have a unique license identification information which can be only decrypted using a proper key.*

"We **consume movies and music through services like Netflix, Spotify, Pandora, Apple Music, etc., often over the web. Netflix, et al. are obligated, due to agreements with content partners, to serve their content with DRM.** Moreover, web streaming is likely a significant chunk of usage for Netflix. This makes **the problem of enforcing DRM over the web an incredibly important problem for Netflix, et al. to solve.**

That's where **EMEs come into play.** They are essentially **a technology that allows the browser to communicate with DRM systems over an encrypted medium. They make it possible for DRM software to work over the browser and relay encrypted content to the user. This means Netflix can serve you a movie that's encrypted and protected by DRM."**

See Exhibit 6, p. 2, "The most controversial HTML5 extension – LogRocket" blog post (Date Accessed 05/01/2018), available at <https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291> (emphasis added)

## CLAIM 1

creating a predetermined key to manipulate the digital signal;

## SPOTIFY

“Say **Felix** is a movie streaming service that serves up movies over the browser and **uses EMEs to protect the content it delivers**. Suppose that user wants to play some encrypted content from Felix. Here’s what happens:

**The browser will load up this content, realize it is encrypted and then trigger some Javascript code.**

This **Javascript will take the encrypted content and pass it along to something called the Content Decryption Module (CDM)**—more on this later.

**The CDM will then trigger a request that will be relayed by Felix to the license server.** This is a Felix-controlled server that determines whether or not a particular user is allowed to play a particular piece of content. **If the license server thinks our user can watch the content, it will send back a decryption key for the content to the CDM.**

**The CDM will then decrypt the content and it will begin playing for the user.”**

The most important **chunk of this whole process is the CDM**. By its nature, the **CDM has to be a piece of code that’s trusted by Felix**. If it isn’t, Felix can’t be sure that it will do what it wants it to. This means that the **CDM has to come with the browser**. There are a couple of different providers for CDMs, including **Google’s Widevine** and Adobe Primetime. Both of these provide the **encryption support necessary for content providers such as Felix to enforce DRM**.

See Exhibit 6, p. 2 & 3, “The most controversial HTML5 extension – LogRocket” blog post (Date Accessed 05/01/2018), available at <https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291> (emphasis added)

## CLAIM 1

creating a predetermined key to manipulate the digital signal;

## SPOTIFY

**“Encrypted Media Extensions provides an API that enables web applications to interact with content protection systems, to allow playback of encrypted audio and video.**

EME is designed to **enable the same app and encrypted files to be used in any browser, regardless of the underlying protection system.** The former is made possible by the standardized APIs and flow while the latter is made possible by the concept of Common Encryption

EME is an extension to the HTMLMediaElement specification — hence the name. Being an 'extension' means that browser support for EME is optional: if a browser does not support encrypted media, it will not be able to play encrypted media, but EME is not required for HTML spec compliance

“EME implementations use the following external components:

**Key System: A content protection (DRM) mechanism. EME doesn't define Key Systems themselves, apart from Clear Key (more about that below).**

**Content Decryption Module (CDM): A client-side software or hardware mechanism that enables playback of encrypted media. As with Key Systems, EME doesn't define any CDMs, but provides an interface for applications to interact with CDMs that are available.**

**License (Key) server: Interacts with a CDM to provide keys to decrypt media. Negotiation with the license server is the responsibility of the application.**

*continued on next slide.*

## CLAIM 1

creating a predetermined key to manipulate the digital signal;

## SPOTIFY

**Packaging service: Encodes and encrypts media for distribution/consumption.**

Note that an application **using EME interacts with a license server to get keys to enable decryption**, but user identity and authentication are not part of EME. **Retrieval of keys to enable media playback happens after (optionally) authenticating a user.** **Services such as Netflix must authenticate users within their web application: when a user signs into the application, the application determines the user's identity and privileges."**

See Exhibit 8, p. 1, "What is EME\_ \_ Web Fundamentals \_ Google Developers" forum page (Date Accessed 05/02/2018), available at <https://developers.google.com/web/fundamentals/media/eme> (emphasis added)

**"The authorization flow we use in this tutorial is the Authorization Code Flow. This flow first gets a code from the Spotify Accounts Service, then exchanges that code for an access token. The code-to-token exchange requires a secret key, and for security is done through direct server-to-server communication."**

See Exhibit 9, p. 1, "Web API Tutorial \_ Spotify for Developers" webpage (Date Accessed 05/02/2018), available at <https://beta.developer.spotify.com/documentation/web-api/quick-start/> (emphasis added)

## CLAIM 1

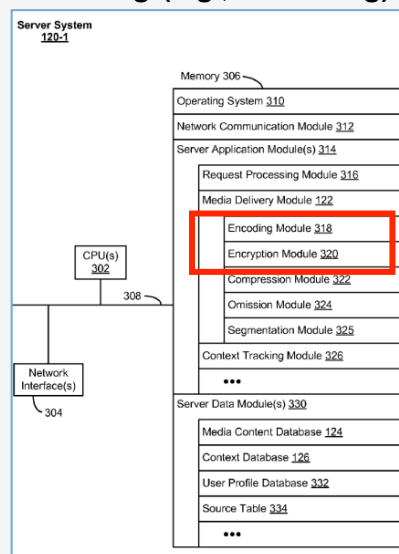
creating a predetermined key to manipulate the digital signal;

## SPOTIFY

“a media delivery module 122 for sending (e.g., streaming) media content to a client device (e.g., client device 110-1) remote from sever system 120-1 in response to the request from client device 110-1, media delivery module 122 including but not limited to:

**an encoding module 318 for encoding media content prior to sending (e.g., streaming) the media content to client device 110-1 or to other content sources for storage;**

**an encryption module 320 for encrypting one or more portions of media content prior to sending (e.g., streaming) the media content to client device 110-1.”**



See Exhibit 30, at Col. 11:08-21 & Fig. 3, “U.S. Patent No. 9,529,888 - System and method for efficiently providing media and associated metadata.” Patent assigned to Spotify (Date Accessed 05/10/2018), available at <https://patentimages.storage.googleapis.com/88/d4/11/6b1fc7c1e09e6b/US9529888.pdf> (emphasis added)

## CLAIM 1

creating a predetermined key to manipulate the digital signal;

## SPOTIFY

*Upon information and belief, Spotify works similarly utilizing the CDM module as explained here.*

### **“Getting a key from a license server**

In typical commercial use, **content will be encrypted and encoded using a packaging service or tool. Once the encrypted media is made available online, a web client can obtain a key (contained within a license) from a license server and use the key to enable decryption and playback of the content.**

The following code (adapted from the spec examples) shows **how an application can select an appropriate key system and obtain a key from a license server.”**

```
var video = document.querySelector('video');

var config = [{initDataTypes: ['webm'],
  videoCapabilities: [{contentType: 'video/webm; codecs="vp09.00.10.08"' }]}];

if (!video.mediaKeys) {
  navigator.requestMediaKeySystemAccess('org.w3.clearkey',
    config).then(
      function(keySystemAccess) {
        var promise = keySystemAccess.createMediaKeys();
        promise.catch(
          console.error.bind(console, 'Unable to create MediaKeys')
        );
        promise.then(
          function(createdMediaKeys) {
            return video.setMediaKeys(createdMediaKeys);
          }
        ).catch(
          console.error.bind(console, 'Unable to set MediaKeys')
        );
        promise.then(
          function(createdMediaKeys) {
            var initData = new Uint8Array([...]);
            var keySession = createdMediaKeys.createSession();
            keySession.addEventListener('message', handleMessage,
              false);
            return keySession.generateRequest('webm', initData);
          }
        ).catch(
          console.error.bind(console,
            'Unable to create or initialize key session')
        );
      }
    );
}

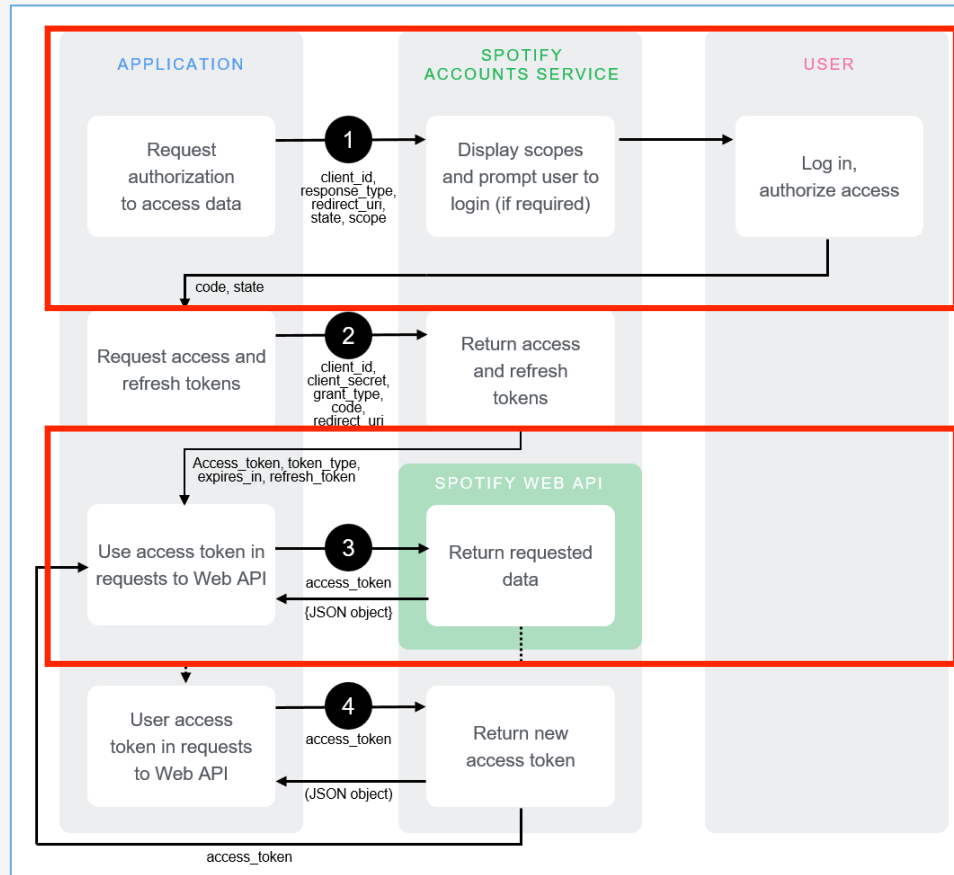
function handleMessage(event) {
  var keySession = event.target;
  var license = new Uint8Array([...]);
  keySession.update(license).catch(
    console.error.bind(console, 'update() failed')
  );
}
```

See Exhibit 8, p. 1, “What is EME \_ \_ Web Fundamentals \_ \_ Google Developers” forum page (Date Accessed 05/02/2018), available at <https://developers.google.com/web/fundamentals/media/eme> (emphasis added).

## CLAIM 1

creating a predetermined  
key to manipulate the  
digital signal;

## SPOTIFY



See Exhibit 11, p. 4, "Authorization Guide \_ Spotify for Developers" webpage (Date Accessed 05/03/2018), available at <https://beta.developer.spotify.com/documentation/general/guides/authorization-guide/> (emphasis added)

## CLAIM 1

creating a predetermined key to manipulate the digital signal;

## SPOTIFY

### 2.3.3 Spotify

Spotify implements a custom protection scheme to prevent duplication of their content. This scheme was reverse-engineered by the Despotify Project in their attempt to create an interoperable client [5]. The scheme uses a stream cipher to protect its communication, and, in addition, it encrypts each individual song.

**Stream cipher.** The Spotify client performs a key exchange with the server to create a key to be used for the remainder of the session. After the key is generated, the session is encrypted using a Shannon stream cipher [42].

**Song encryption.** Individual music files sent by Spotify in the encrypted stream are themselves encrypted with AES. The keys to this encryption are sent in the stream along with the music files. Upon receipt of a music file and its corresponding key, the Spotify client decrypts the file for playback. For offline playback, Spotify can cache this data.

**Performance.** An online analysis of Spotify must be fast enough to process the data stream without dropping the connection. Additionally, if the Spotify client runs too slow, it will mistakenly perceive that the connection to the server has been lost.

See Exhibit 32, p. 5, “steal-this-movie-automatically-bypassing-drm-protection-usenix2013” PDF (Date Accessed 05 / 15 / 2018), available at <http://sefcom.asu.edu/publications/steal-this-movie-automatically-bypassing-drm-protection-usenix2013.pdf> (emphasis added)



## CLAIM 1

creating a predetermined  
key to manipulate the  
digital signal;

## SPOTIFY

### “How does EME Work?”

1. A web application attempts to play audio or video that has one or more encrypted streams.
2. The browser recognizes that the media is encrypted (see box below for how that happens) and fires an encrypted event with metadata (initData) obtained from the media about the encryption.
3. The application handles the encrypted event.
  - a. If no MediaKeys object has been associated with the media element, first select an available Key System by using navigator.requestMediaKeySystemAccess() to check what Key Systems are available, then create a MediaKeys object for an available Key System via a MediaKeySystemAccess object. Note that initialization of the MediaKeys object should happen before the first encrypted event. Getting a license server URL is done by the app independently of selecting an available key system. A MediaKeys object represents all the keys available to decrypt the media for an audio or video element. It represents a CDM instance and provides access to the CDM, specifically for creating key sessions, which are used to obtain keys from a license server.
  - b. Once the MediaKeys object has been created, assign it to the media element: setMediaKeys() associates the MediaKeys object with an HTMLMediaElement, so that its keys can be used during playback, i.e. during decoding.
4. The app creates a MediaKeySession by calling createSession() on the MediaKeys. This creates a MediaKeySession, which represents the lifetime of a license and its key(s).
5. The app generates a license request by passing the media data obtained in the encrypted handler to the CDM by calling generateRequest() on the MediaKeySession.
6. The CDM fires a message event: a request to acquire a key from a license server.
7. The MediaKeySession object receives the message event and the application sends a message to the license server (via XHR, for example).
8. The application receives a response from the license server and passes the data to the CDM using the update() method of the MediaKeySession.
9. The CDM decrypts the media using the keys in the license. A valid key may be used, from any session within the MediaKeys associated with the media element. The CDM will access the key and policy, indexed by Key ID.

Media playback resumes.

How does the browser know that media is encrypted?

This information is in the metadata of the media container file, which will be in a format such as ISO BMFF or WebM. For ISO BMFF this means header metadata, called the protection scheme information box. WebM uses the Matroska ContentEncryption element, with some WebM-specific additions. Guidelines are provided for each container in an EME-specific registry.

Note that there may be multiple messages between the CDM and the license server, and all communication in this process is opaque to the browser and application: messages are only understood by the CDM and license server, although the app layer can see what type of message (https://w3c.github.io/encrypted-media/#idl-def-MediaKeyMessageType) the CDM is sending. The license request contains proof of the CDM's validity (and trust relationship) as well as a key to use when encrypting the content key(s) in the resulting license.

See Exhibit 8, p. 1, “What is EME\_ \_ Web Fundamentals \_ Google Developers” forum page (Date Accessed 05/02/2018), available at <https://developers.google.com/web/fundamentals/media/eme> (emphasis added)

*Thus, this claim element is infringed literally, or in the alternative under the doctrine of equivalents.*

## CLAIM 1

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

## SPOTIFY

*The data is manipulated (through **Encrypted Media Extensions (EME)**) before transmission from Party 1 to Party 2 so that it is not compromised or deciphered by an unauthorized user. The package's encrypted content will have a unique license identification information which can be only decrypted using a proper key. Without the key, the music is encoded at a lower bitrate (parametrized by the file format information) than if accessed through Spotify Premium.*

"We **consume movies and music through services like Netflix, Spotify, Pandora, Apple Music, etc., often over the web. Netflix, et al. are obligated, due to agreements with content partners, to serve their content with DRM.** Moreover, web streaming is likely a significant chunk of usage for Netflix. This makes **the problem of enforcing DRM over the web an incredibly important problem for Netflix, et al. to solve.**

That's where **EMEs come into play.** They are essentially **a technology that allows the browser to communicate with DRM systems over an encrypted medium.** They **make it possible for DRM software to work over the browser and relay encrypted content to the user.** This means **Netflix can serve you a movie that's encrypted and protected by DRM."**

See Exhibit 6, p. 2, "The most controversial HTML5 extension – LogRocket" blog post (Date Accessed 05/01/2018), available at <https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291> (emphasis added).

## CLAIM 1

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

## SPOTIFY

**“Say Felix is a movie streaming service that serves up movies over the browser and uses EMEs to protect the content it delivers.** Suppose that user wants to play some encrypted content from Felix. Here’s what happens:

**The browser will load up this content, realize it is encrypted and then trigger some Javascript code.**

This Javascript will take the encrypted content and pass it along to something called **the Content Decryption Module (CDM)**—more on this later.

**The CDM will then trigger a request that will be relayed by Felix to the license server.** This is a Felix-controlled server that determines whether or not a particular user is allowed to play a particular piece of content. **If the license server thinks our user can watch the content, it will send back a decryption key for the content to the CDM.**

**The CDM will then decrypt the content and it will begin playing for the user.”**

The most important **chunk of this whole process is the CDM.** By its nature, the **CDM has to be a piece of code that’s trusted by Felix.** If it isn’t, Felix can’t be sure that it will do what it wants it to. This means that the **CDM has to come with the browser.** There are a couple of different providers for CDMs, including **Google’s Widevine** and Adobe Primetime. Both of these provide the **encryption support necessary for content providers such as Felix to enforce DRM.**

See Exhibit 6, p. 2 & 3, “The most controversial HTML5 extension – LogRocket” blog post (Date Accessed 05/01/2018), available at <https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291> (emphasis added).

## CLAIM 1

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

## SPOTIFY

**“Encrypted Media Extensions provides an API that enables web applications to interact with content protection systems, to allow playback of encrypted audio and video.**

EME is designed to **enable the same app and encrypted files to be used in any browser, regardless of the underlying protection system.** The former is made possible by the standardized APIs and flow while the latter is made possible by the concept of Common Encryption

EME is an extension to the HTMLMediaElement specification — hence the name. Being an 'extension' means that browser support for EME is optional: if a browser does not support encrypted media, it will not be able to play encrypted media, but EME is not required for HTML spec compliance

“EME implementations use the following external components:

**Key System: A content protection (DRM) mechanism. EME doesn't define Key Systems themselves, apart from Clear Key (more about that below).**

**Content Decryption Module (CDM): A client-side software or hardware mechanism that enables playback of encrypted media. As with Key Systems, EME doesn't define any CDMs, but provides an interface for applications to interact with CDMs that are available.**

**License (Key) server: Interacts with a CDM to provide keys to decrypt media. Negotiation with the license server is the responsibility of the application.**

*continued on next slide.*

## CLAIM 1

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

## SPOTIFY

**Packaging service: Encodes and encrypts media for distribution/consumption.**

Note that an application **using EME interacts with a license server to get keys to enable decryption**, but user identity and authentication are not part of EME. **Retrieval of keys to enable media playback happens after (optionally) authenticating a user. Services such as Netflix must authenticate users within their web application: when a user signs into the application, the application determines the user's identity and privileges."**

See Exhibit 8, p. 1, "What is EME\_ \_ Web Fundamentals \_ Google Developers" forum page (Date Accessed 05/02/2018), available at <https://developers.google.com/web/fundamentals/media/eme> (emphasis added).

**"The authorization flow we use in this tutorial is the Authorization Code Flow. This flow first gets a code from the Spotify Accounts Service, then exchanges that code for an access token. The code-to-token exchange requires a secret key, and for security is done through direct server-to-server communication."**

See Exhibit 9, p. 1, "Web API Tutorial \_ Spotify for Developers" webpage (Date Accessed 05/02/2018), available at <https://beta.developer.spotify.com/documentation/web-api/quick-start/> (emphasis added).

## CLAIM 1

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

## SPOTIFY

### “How does EME Work?”

1. A web application attempts to play audio or video that has one or more encrypted streams.
2. The browser recognizes that the media is encrypted (see box below for how that happens) and fires an encrypted event with metadata (initData) obtained from the media about the encryption.
3. The application handles the encrypted event:
  - a. If no MediaKeys object has been associated with the media element, first select an available Key System by using navigator.requestMediaKeySystemAccess() to check what Key Systems are available, then create a MediaKeys object for an available Key System via a MediaKeySystemAccess object. Note that initialization of the MediaKeys object should happen before the first encrypted event. Getting a license server URL is done by the app independently of selecting an available key system. A MediaKeys object represents all the keys available to decrypt the media for an audio or video element. It represents a CDM instance and provides access to the CDM, specifically for creating key sessions, which are used to obtain keys from a license server.
  - b. Once the MediaKeys object has been created, assign it to the media element: setMediaKeys() associates the MediaKeys object with an HTMLMediaElement, so that its keys can be used during playback, i.e. during decoding.
4. The app creates a MediaKeySession by calling createSession() on the MediaKeys. This creates a MediaKeySession, which represents the lifetime of a license and its key(s).
5. The app generates a license request by passing the media data obtained in the encrypted handler to the CDM by calling generateRequest() on the MediaKeySession.
6. The CDM fires a message event: a request to acquire a key from a license server.
7. The MediaKeySession object receives the message event and the application sends a message to the license server (via XHR, for example).
8. The application receives a response from the license server and passes the data to the CDM using the update() method of the MediaKeySession.
9. The CDM decrypts the media using the keys in the license. A valid key may be used, from any session within the MediaKeys associated with the media element. The CDM will access the key and policy, indexed by Key ID.

Media playback resumes.

How does the browser know that media is encrypted?

This information is in the metadata of the media container file, which will be in a format such as ISO BMFF or WebM. For ISO BMFF this means header metadata, called the protection scheme information box. WebM uses the Matroska ContentEncryption element, with some WebM-specific additions. Guidelines are provided for each container in an EME-specific registry.

Note that there may be multiple messages between the CDM and the license server, and all communication in this process is opaque to the browser and application: messages are only understood by the CDM and license server, although the app layer can see what type of message (https://w3c.github.io/encrypted-media/#idl-def-MediaKeyMessageType) the CDM is sending. The license request contains proof of the CDM's validity (and trust relationship) as well as a key to use when encrypting the content key(s) in the resulting license.

See Exhibit 8, p. 1, “What is EME \_ \_ Web Fundamentals \_ \_ Google Developers” forum page (Date Accessed 05/02/2018), available at <https://developers.google.com/web/fundamentals/media/eme> (emphasis added).

## CLAIM 1

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

## SPOTIFY

*Upon information and belief, Spotify works similarly utilizing the CDM module as explained here.*

### **“Getting a key from a license server**

In typical commercial use, **content will be encrypted and encoded using a packaging service or tool. Once the encrypted media is made available online, a web client can obtain a key (contained within a license) from a license server and use the key to enable decryption and playback of the content.**

The following code (adapted from the spec examples) shows **how an application can select an appropriate key system and obtain a key from a license server.”**

```
var video = document.querySelector('video');
var config = [{initDataTypes: ['webm'],
  videoCapabilities: [{contentType: 'video/webm; codecs="vp09.00.10.08"' }]}];
if (!video.mediaKeys) {
  navigator.requestMediaKeySystemAccess('org.w3.clearkey',
    config).then(
      function(keySystemAccess) {
        var promise = keySystemAccess.createMediaKeys();
        promise.catch(
          console.error.bind(console, 'Unable to create MediaKeys')
        );
        promise.then(
          function(createdMediaKeys) {
            return video.setMediaKeys(createdMediaKeys);
          }
        ).catch(
          console.error.bind(console, 'Unable to set MediaKeys')
        );
        promise.then(
          function(createdMediaKeys) {
            var initData = new Uint8Array([...]);
            var keySession = createdMediaKeys.createSession();
            keySession.addEventListener('message', handleMessage,
              false);
            return keySession.generateRequest('webm', initData);
          }
        ).catch(
          console.error.bind(console,
            'Unable to create or initialize key session')
        );
      }
    );
}
function handleMessage(event) {
  var keySession = event.target;
  var license = new Uint8Array([...]);
  keySession.update(license).catch(
    console.error.bind(console, 'update() failed')
  );
}
```

See Exhibit 8, p. 1, “What is EME\_ \_ Web Fundamentals \_ Google Developers” forum page (Date Accessed 05/02/2018), available at <https://developers.google.com/web/fundamentals/media/eme> (emphasis added).

## CLAIM 1

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

## SPOTIFY

### 2.3.3 Spotify

Spotify implements a custom protection scheme to prevent duplication of their content. This scheme was reverse-engineered by the Despotify Project in their attempt to create an interoperable client [5]. The scheme uses a stream cipher to protect its communication, and, in addition, it encrypts each individual song.

**Stream cipher.** The Spotify client performs a key exchange with the server to create a key to be used for the remainder of the session. After the key is generated, the session is encrypted using a Shannon stream cipher [42].

**Song encryption.** Individual music files sent by Spotify in the encrypted stream are themselves encrypted with AES. The keys to this encryption are sent in the stream along with the music files. Upon receipt of a music file and its corresponding key, the Spotify client decrypts the file for playback. For offline playback, Spotify can cache this data.

**Performance.** An online analysis of Spotify must be fast enough to process the data stream without dropping the connection. Additionally, if the Spotify client runs too slow, it will mistakenly perceive that the connection to the server has been lost.

See Exhibit 32, p. 5, “steal-this-movie-automatically-bypassing-drm-protection-usenix2013” PDF (Date Accessed 05 / 15 / 2018), available at <http://sefcom.asu.edu/publications/steal-this-movie-automatically-bypassing-drm-protection-usenix2013.pdf> (emphasis added)



## CLAIM 1

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

## SPOTIFY

“There are **two basic ways you can authenticate your application’s user and pass the scopes needed to get authorization to access user data:**

The preferred and highly recommended way: **Use the Spotify client to authenticate the user (with a fallback to login through a WebView).** If Spotify is installed on the device, **SDK connects to the Spotify client and uses current session.** If Spotify is not installed, this method will fall back to using **Android WebView class which allows you to display a web page as part of your activity layout.** **Authentication and authorization takes place in the WebView without leaving the application.**

Only if the first way is not possible: **Login through a web browser. This method opens the Spotify Accounts page in a separate, external browser window, completes the authentication process, and then redirects back to your application.”**

See Exhibit 10, p. 1, “Android SDK Authentication Guide \_ Spotify for Developers” webpage (Date Accessed 05/02/2018), available at <https://beta.developer.spotify.com/documentation/android-sdk/guides/android-authentication/> (emphasis added).

## CLAIM 1

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

## SPOTIFY

“Single **Sign-On with Spotify Client** and a WebView Fallback

In this flow, **the Android SDK will try to fetch the authorization code/access token using the Spotify Android client.**

Note: to be able to use Single Sign-On you need to register your application’s fingerprint. Please see Registering Application Fingerprint section of the tutorial.

**If Spotify is installed on the device, the SDK will connect to the Spotify client and fetch the authorization code/access token for current user.** Since the user is already logged into Spotify, they don’t need to type in their username and password. If the SDK application requests scopes that have not been approved before, the user will see a list of scopes and will need to accept them.

**If Spotify is not installed on the device, the SDK will fallback to the WebView based authorization and open the Spotify Accounts login page at https://accounts.spotify.com in a native WebView.** User will have to enter their username and password to login to Spotify and accept the supplied scopes.

**In both cases the result of the authorization flow will be returned in the onActivityResult method of the activity that initiated it.**

This flow is entirely completed within the application; there is no need to open a web browser.”

See Exhibit 10, p. 2, “Android SDK Authentication Guide \_ Spotify for Developers” webpage (Date Accessed 05/02/2018), available at <https://beta.developer.spotify.com/documentation/android-sdk/guides/android-authentication/> (emphasis added).

## CLAIM 1

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

## SPOTIFY

### **“6. Audio Quality: 160Kbps VS 320Kbps**

On desktop, **default streaming audio quality for Spotify Free users is Ogg Vorbis 160kbit/s while Premium subscribers can choose to switch on High quality streaming, which uses 320kbit/s.** On iPhone, iPad or Android, Spotify Free users can choose either 96kbit/s or 160kbit/s while Premium subscribers have one more option which is 320kbit/s. As to streaming audio quality on Chromecast, it's AAC 128kbit/s for Spotify Free, and 256kbit/s for Spotify Premium.”

See Exhibit 27, p. 5, “Spotify Free VS Premium\_ 6 Differences You Need to Know” webpage (Date Accessed 05/08/2018), available at <http://www.tunemobie.com/resources/spotify-free-vs-premium-6-differences.html> (emphasis added)

### **“AUDIO QUALITY**

**Spotify uses three different quality settings for streaming, all in the Ogg Vorbis format. 96kbps is the standard bitrate for mobile, which then jumps to 160kbps for desktop and web player 'standard quality' and 'high quality' on mobile.**

**If you pay the monthly fee for Spotify Premium, you'll get 320kbps which is 'high quality' on desktop and 'extreme quality' on mobile.”**

See Exhibit 28, p. 3, “Spotify Free vs Spotify Premium - Tech Advisor” webpage (Date Accessed 05/08/2018), available at <https://www.techadvisor.co.uk/review/audio-and-music-software/spotify-free-vs-spotify-premium-3597766/> (emphasis added)

## CLAIM 1

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

## SPOTIFY

*Spotify complies with RFC 6749 standard for authorization code flow*

### 4.1. Authorization Code Grant

The authorization code grant type is used to obtain both access tokens and refresh tokens and is optimized for confidential clients. Since this is a redirection-based flow, the client must be capable of interacting with the resource owner's user-agent (typically a web browser) and capable of receiving incoming requests (via redirection) from the authorization server.

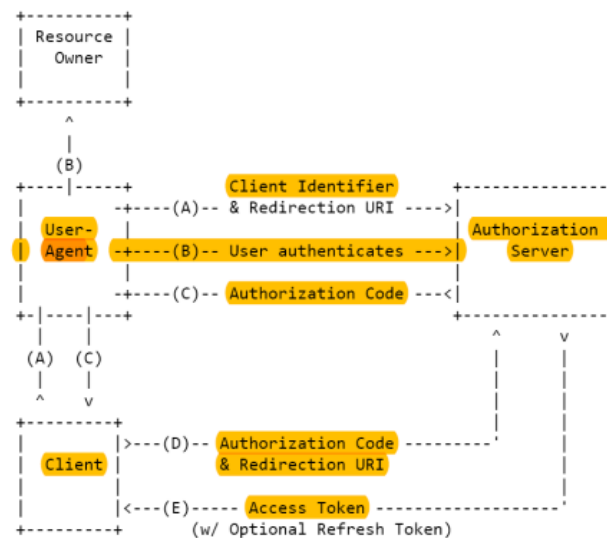


Figure 3: Authorization Code Flow

See Exhibit 14, p. 24, "RFC 6749 - The OAuth 2.0 Authorization Framework" webpage (Date Accessed 05/01/2018), available at <https://tools.ietf.org/html/rfc6749#section-4.1> (emphasis added)

## CLAIM 1

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

## SPOTIFY

“The flow illustrated in Figure 3 includes the following steps:

- (A) The **client initiates the flow by directing the resource owner's user-agent to the authorization endpoint**. The **client includes its client identifier, requested scope, local state, and a redirection URI to which the authorization server will send the user-agent back once access is granted** (or denied).
- (B) The **authorization server authenticates the resource owner** (via the user-agent) and establishes whether the resource owner grants or denies the client's access request.
- (C) Assuming the resource owner grants access, **the authorization server redirects the user-agent back to the client using the redirection URI provided earlier** (in the request or during client registration). The redirection URI includes an authorization code and any local state provided by the client earlier.
- (D) The **client requests an access token from the authorization server's token endpoint by including the authorization code received in the previous step**. When making the request, the client authenticates with the authorization server. **The client includes the redirection URI used to obtain the authorization code for verification**.
- (E) The **authorization server authenticates the client, validates the authorization code, and ensures that the redirection URI received matches the URI used to redirect the client in step (C)**. If valid, **the authorization server responds back with an access token** and, optionally, a refresh token.”

See Exhibit 14, p. 25, “RFC 6749 - The OAuth 2.0 Authorization Framework” webpage (Date Accessed 05/01/2018), available at <https://tools.ietf.org/html/rfc6749#section-4.1> (emphasis added)

## CLAIM 1

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

## SPOTIFY

### Obtaining Authorization

Making authorized requests to the Spotify platform requires that you are granted permission to access data.

In accordance with RFC-6750, 3 parties are involved in the authorization process:

- Server: the Spotify server
- Client: your application
- Resource: the end user data and controls



### To Obtain Authorization:

1. Register your application.
2. Follow one of the 3 Spotify authorization flows.

### Authorization Flows

There are 3 optional flows to obtaining app authorization:

- Refreshable user authorization: **Authorization Code**
- Temporary user authorization: **Implicit Grant**
- Refreshable app authorization: **Client Credentials Flow**

FLOW	ACCESS USER RESOURCES	ACCESS TOKEN REFRESH	IMPROVED RATE LIMITS
<b>Authorization Code</b>	Yes	Yes	Yes
Client Credentials	No	No	Yes
Implicit Grant	Yes	No	Yes

For further information and examples of these flows, read our step-by-step [tutorial](#). In addition, see a list of handy [wrappers and tools](#) for your language of choice.

### Authorization Code Flow

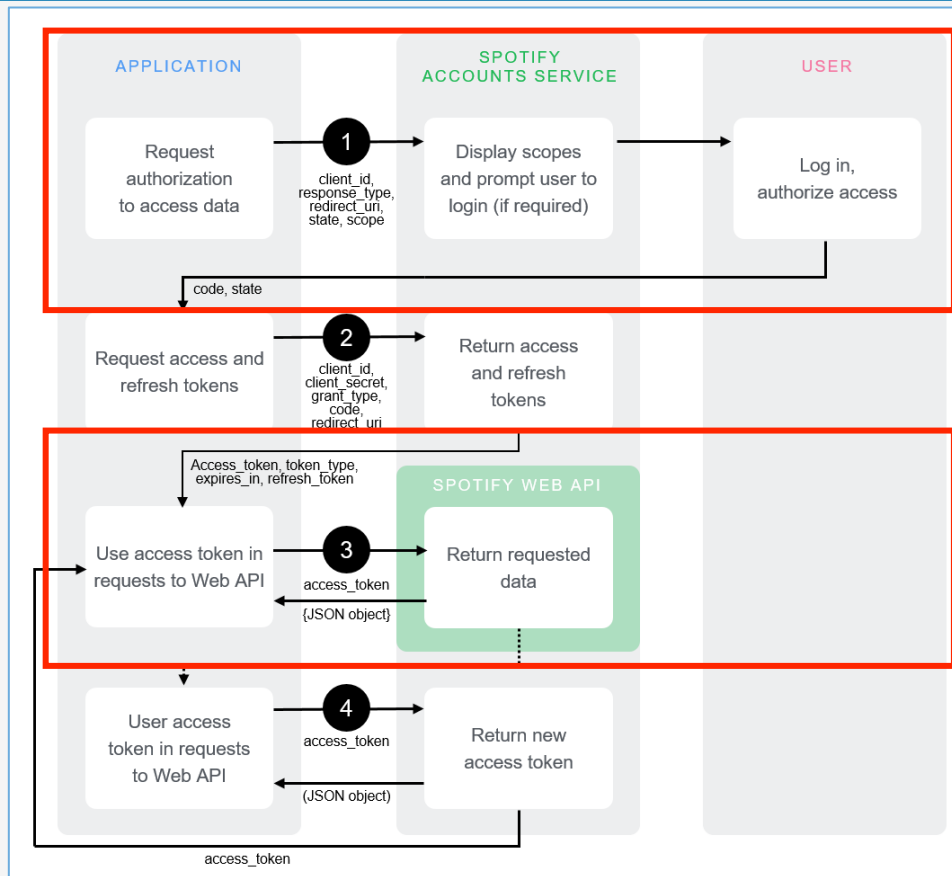
This flow is suitable for long-running applications in which the user grants permission only once. It provides an **access token** that can be refreshed. Since the token exchange involves sending your secret key, perform this on a secure location, like a backend service, and not from a client such as a browser or from a mobile app.

See Exhibit 11, p. 1-3, "Authorization Guide \_ Spotify for Developers" webpage (Date Accessed 05 / 03 / 2018), available at <https://beta.developer.spotify.com/documentation/general/guides/authorization-guide/> (emphasis added).

## CLAIM 1

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

## SPOTIFY



See Exhibit 11, p. 4, "Authorization Guide \_ Spotify for Developers" webpage (Date Accessed 05/03/2018), available at <https://beta.developer.spotify.com/documentation/general/guides/authorization-guide/> (emphasis added)

## CLAIM 1

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

## SPOTIFY

**“App creates one or more key sessions associated with the selected DRM system, each of which will manage one or more decryption keys (licenses)**

The app instructs the key session to generate a license request message by providing it with initialization data. The browser may provide this data by means of the event in Step 1, or it may be acquired by the app through other means (i.e. in an ABR manifest file).

The CDM for the selected DRM system will generate a data blob (license request) and deliver it to the app.

The app sends the license request to a license server.

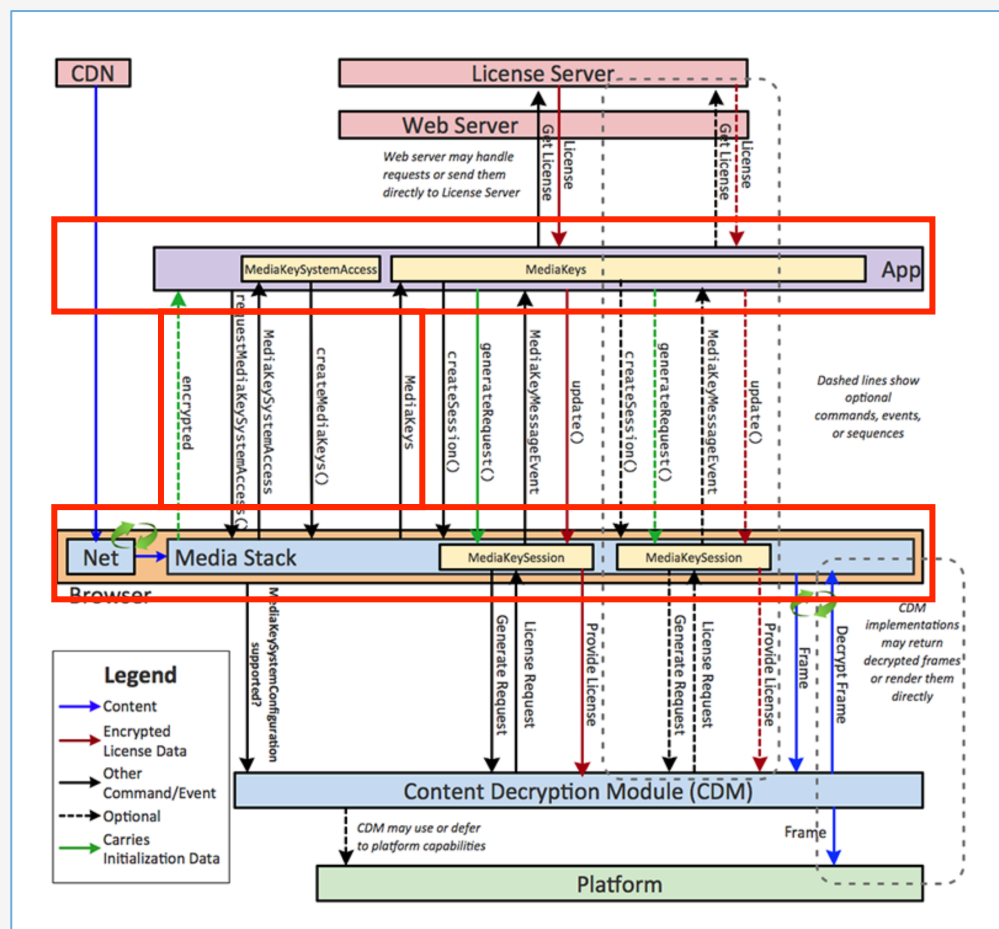
Upon receiving a response to its license request, the app passes the response message back to the CDM. The CDM adds to the key session any decryption keys contained within the response.”

See Exhibit 31, p. 2, “Encrypted Media Extensions Provide a Common Ground - CableLabs” webpage (Date Accessed 05/14/2018), available at <https://www.cablelabs.com/encrypted-media-extensions-provide-a-common-ground//> (emphasis added)



# SPOTIFY

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

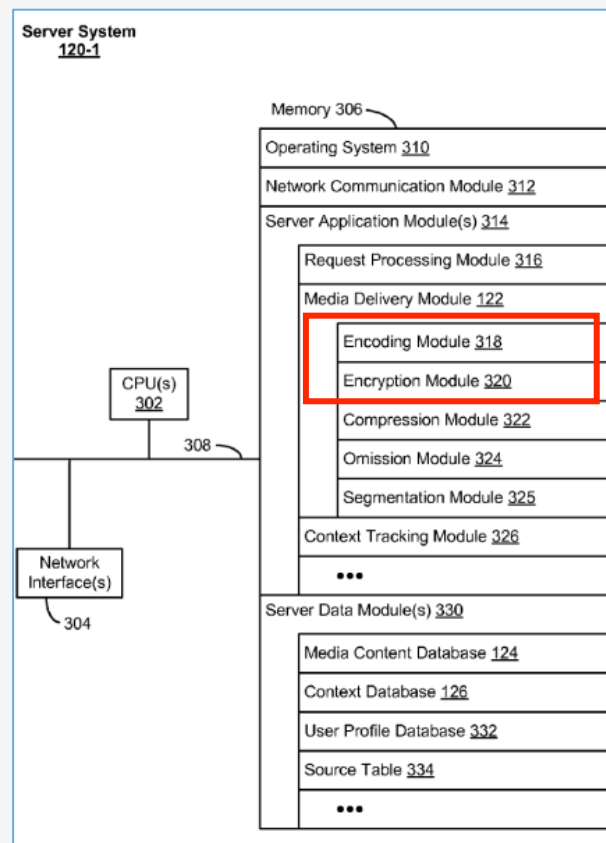


See Exhibit 31, p. 1, “Encrypted Media Extensions Provide a Common Ground - CableLabs” webpage (Date Accessed 05/14/2018), available at <https://www.cablelabs.com/encrypted-media-extensions-provide-a-common-ground//> (emphasis added)

## CLAIM 1

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

## SPOTIFY



See Exhibit 30, at Fig. 3, "U.S. Patent No. 9,529,888 - System and method for efficiently providing media and associated metadata." Patent assigned to Spotify (Date Accessed 05/10/2018), available at <https://patentimages.storage.googleapis.com/88/d4/11/6b1fc7c1e09e6b/US9529888.pdf> (emphasis added)

## CLAIM 1

manipulating the digital signal using the predetermined key to generate at least one permutation of the digital signal parameterized by the file format information defining how the digital signal is encoded;

## SPOTIFY

**“a media delivery module 122 for sending (e.g., streaming) media content to a client device (e.g., client device 110-1) remote from sever system 120-1 in response to the request from client device 110-1, media delivery module 122 including but not limited to:**

**an encoding module 318 for encoding media content prior to sending (e.g., streaming) the media content to client device 110-1 or to other content sources for storage;**

**an encryption module 320 for encrypting one or more portions of media content prior to sending (e.g., streaming) the media content to client device 110-1.**

**a compression module 322 for compressing media content prior to sending (e.g., streaming) the media content to client device 110-1;**

**an omission module 324 for omitting information from media content prior sending (e.g., streaming) the media content to client device 110-1 (e.g., by determining redundant information that can be omitted when compressing the media content while maintaining a quality of the media content above a predefined quality level); and”**

See Exhibit 30, at Col. 11:08-32, “U.S. Patent No. 9,529,888 - System and method for efficiently providing media and associated metadata.” Patent assigned to Spotify (Date Accessed 05/10/2018), available at <https://patentimages.storage.googleapis.com/88/d4/11/6b1fc7c1e09e6b/US9529888.pdf> (emphasis added)

*Thus, this claim element is infringed literally, or in the alternative under the doctrine of equivalents.*

## CLAIM 1

wherein the predetermined key comprises one or more mask sets having random or pseudo-random series of bits; and

## SPOTIFY

*Upon information and belief, Spotify uses Shannon Stream Cipher for encrypting the music stream which is created using a pseudorandom keystream bits.*

### 2.3.3 Spotify

Spotify implements a custom protection scheme to prevent duplication of their content. This scheme was reverse-engineered by the Despotify Project in their attempt to create an interoperable client [5]. The scheme uses a stream cipher to protect its communication, and, in addition, it encrypts each individual song.

**Stream cipher.** The Spotify client performs a key exchange with the server to create a key to be used for the remainder of the session. After the key is generated, the session is encrypted using a Shannon stream cipher [42].

**Song encryption.** Individual music files sent by Spotify in the encrypted stream are themselves encrypted with AES. The keys to this encryption are sent in the stream along with the music files. Upon receipt of a music file and its corresponding key, the Spotify client decrypts the file for playback. For offline playback, Spotify can cache this data.

**Performance.** An online analysis of Spotify must be fast enough to process the data stream without dropping the connection. Additionally, if the Spotify client runs too slow, it will mistakenly perceive that the connection to the server has been lost.

See Exhibit 32, p. 5, “steal-this-movie-automatically-bypassing-drm-protection-usenix2013” PDF (Date Accessed 05 / 15 / 2018), available at <http://sefcom.asu.edu/publications/steal-this-movie-automatically-bypassing-drm-protection-usenix2013.pdf> (emphasis added)

## CLAIM 1

wherein the predetermined key comprises one or more mask sets having random or pseudo-random series of bits; and

## SPOTIFY

“Stream ciphers can be viewed as approximating the action of a proven unbreakable cipher, the one-time pad (OTP), sometimes known as the Vernam cipher. **A one-time pad uses a keystream of completely random digits. The keystream is combined with the plaintext digits one at a time to form the ciphertext. This system was proved to be secure by Claude E. Shannon in 1949.** However, the keystream must be generated completely at random with at least the same length as the plaintext and cannot be used more than once.”

See Exhibit 33, p. 1, “Stream cipher - Wikipedia “ Wikipedia article (Date Accessed 05/15/2018), available at [https://en.wikipedia.org/wiki/Stream\\_cipher](https://en.wikipedia.org/wiki/Stream_cipher) (emphasis added)

### shannon

build passing go report A+

Pure Go implementation of Shannon stream cipher. No-brainer port of [rust-shannon](#).  
Shannon cipher is used in Spotify Connect to encrypt communication between player and Spotify AP server. Shannon cipher is variant of [Sober](#) stream cipher.

### Example

Encryption

```
import "github.com/chatoooo/shannon"
...
key := []byte{0x65, 0x87, 0xd8, 0x8f, 0x6c, 0x32, 0x9d, 0x8a, 0xe4, 0x6b}
message := []byte("My secret message")
cipher := shannon.New(key)
cipher.Encrypt(message)
// message contains ciphertext now
mac := make([]byte, 16)
cipher.Final(mac)
// mac contains MAC of the message
```

See Exhibit 34, p. 1, “GitHub - chatoooo\_shannon\_ Pure Go implementation of shannon stream cipher“ Wikipedia article (Date Accessed 05/15/2018), available at <https://github.com/chatoooo/shannon>

## CLAIM 1

wherein the predetermined key comprises one or more mask sets having random or pseudo-random series of bits; and

## SPOTIFY

**“Say Felix is a movie streaming service that serves up movies over the browser and uses EMEs to protect the content it delivers.** Suppose that user wants to play some encrypted content from Felix. Here’s what happens:

**The browser will load up this content, realize it is encrypted and then trigger some Javascript code.**

This Javascript will take the encrypted content and pass it along to something called the **Content Decryption Module (CDM)**—more on this later.

**The CDM will then trigger a request that will be relayed by Felix to the license server.** This is a Felix-controlled server that determines whether or not a particular user is allowed to play a particular piece of content. **If the license server thinks our user can watch the content, it will send back a decryption key for the content to the CDM.**

**The CDM will then decrypt the content and it will begin playing for the user.”**

The most important **chunk of this whole process is the CDM.** By its nature, the **CDM has to be a piece of code that’s trusted by Felix.** If it isn’t, Felix can’t be sure that it will do what it wants it to. This means that the **CDM has to come with the browser.** There are a couple of different providers for CDMs, including **Google’s Widevine** and Adobe Primetime. Both of these provide the **encryption support necessary for content providers such as Felix to enforce DRM.**

See Exhibit 6, p. 2 & 3, “The most controversial HTML5 extension – LogRocket” blog post (Date Accessed 05/01/2018), available at <https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291> (emphasis added)

## CLAIM 1

wherein the predetermined key comprises one or more mask sets having random or pseudo-random series of bits; and

## SPOTIFY

**“Encrypted Media Extensions provides an API that enables web applications to interact with content protection systems, to allow playback of encrypted audio and video.**

EME is designed to **enable the same app and encrypted files to be used in any browser, regardless of the underlying protection system.** The former is made possible by the standardized APIs and flow while the latter is made possible by the concept of Common Encryption

EME is an extension to the HTMLMediaElement specification — hence the name. Being an 'extension' means that browser support for EME is optional: if a browser does not support encrypted media, it will not be able to play encrypted media, but EME is not required for HTML spec compliance

“EME implementations use the following external components:

**Key System: A content protection (DRM) mechanism. EME doesn't define Key Systems themselves, apart from Clear Key (more about that below).**

**Content Decryption Module (CDM): A client-side software or hardware mechanism that enables playback of encrypted media. As with Key Systems, EME doesn't define any CDMs, but provides an interface for applications to interact with CDMs that are available.**

**License (Key) server: Interacts with a CDM to provide keys to decrypt media. Negotiation with the license server is the responsibility of the application.**

*continued on next slide.*

## CLAIM 1

wherein the predetermined key comprises one or more mask sets having random or pseudo-random series of bits; and

## SPOTIFY

**Packaging service: Encodes and encrypts media for distribution/consumption.**

Note that an application **using EME interacts with a license server to get keys to enable decryption**, but user identity and authentication are not part of EME. **Retrieval of keys to enable media playback happens after (optionally) authenticating a user.** **Services such as Netflix must authenticate users within their web application: when a user signs into the application, the application determines the user's identity and privileges."**

See Exhibit 8, p. 1, "What is EME\_ \_ Web Fundamentals \_ Google Developers" forum page (Date Accessed 05/02/2018), available at <https://developers.google.com/web/fundamentals/media/eme> (emphasis added)

**"The authorization flow we use in this tutorial is the Authorization Code Flow. This flow first gets a code from the Spotify Accounts Service, then exchanges that code for an access token. The code-to-token exchange requires a secret key, and for security is done through direct server-to-server communication."**

See Exhibit 9, p. 1, "Web API Tutorial \_ Spotify for Developers" webpage (Date Accessed 05/02/2018), available at <https://beta.developer.spotify.com/documentation/web-api/quick-start/> (emphasis added)



## CLAIM 1

wherein the predetermined key comprises one or more mask sets having random or pseudo-random series of bits; and

## SPOTIFY

**“a media delivery module 122 for sending (e.g., streaming) media content to a client device (e.g., client device 110-1) remote from sever system 120-1 in response to the request from client device 110-1, media delivery module 122 including but not limited to:**

**an encoding module 318 for encoding media content prior to sending (e.g., streaming) the media content to client device 110-1 or to other content sources for storage;**

**an encryption module 320 for encrypting one or more portions of media content prior to sending (e.g., streaming) the media content to client device 110-1.**

**a compression module 322 for compressing media content prior to sending (e.g., streaming) the media content to client device 110-1;**

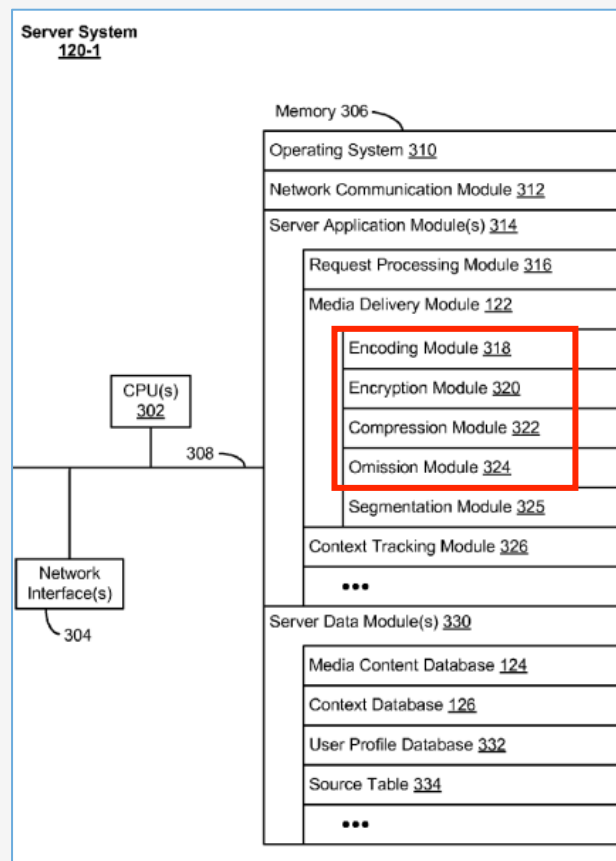
**an omission module 324 for omitting information from media content prior sending (e.g., streaming) the media content to client device 110-1 (e.g., by determining redundant information that can be omitted when compressing the media content while maintaining a quality of the media content above a predefined quality level); and”**

See Exhibit 30, at Col. 11:08-32, “U.S. Patent No. 9,529,888 - System and method for efficiently providing media and associated metadata.” Patent assigned to Spotify (Date Accessed 05/10/2018), available at <https://patentimages.storage.googleapis.com/88/d4/11/6b1fc7c1e09e6b/US9529888.pdf> (emphasis added)

## CLAIM 1

wherein the predetermined key comprises one or more mask sets having random or pseudo-random series of bits; and

## SPOTIFY



See Exhibit 30, at Col. 11:08-32 & Fig. 3, “U.S. Patent No. 9,529,888 - System and method for efficiently providing media and associated metadata.” Patent assigned to Spotify (Date Accessed 05/10/2018), available at <https://patentimages.storage.googleapis.com/88/d4/11/6b1fc7c1e09e6b/US9529888.pdf> (emphasis added)

*Thus, this claim element is infringed literally, or in the alternative under the doctrine of equivalents.*

## CLAIM 1

validating the one or more mask sets either before or after manipulating the file format information using the predetermined key.

## SPOTIFY

*Upon information and belief, the music is decrypted at the user's end by using the same key used to manipulate the digital signal.*

**Packaging service: Encodes and encrypts media for distribution/consumption.**

Note that an application **using EME interacts with a license server to get keys to enable decryption**, but user identity and authentication are not part of EME. **Retrieval of keys to enable media playback happens after (optionally) authenticating a user. Services such as Netflix must authenticate users within their web application: when a user signs into the application, the application determines the user's identity and privileges."**

See Exhibit 8, p. 1, "What is EME \_ Web Fundamentals \_ Google Developers" forum page (Date Accessed 05/02/2018), available at <https://developers.google.com/web/fundamentals/media/eme> (emphasis added)

**"The authorization flow we use in this tutorial is the Authorization Code Flow. This flow first gets a code from the Spotify Accounts Service, then exchanges that code for an access token. The code-to-token exchange requires a secret key, and for security is done through direct server-to-server communication."**

See Exhibit 9, p. 1, "Web API Tutorial \_ Spotify for Developers" webpage (Date Accessed 05/02/2018), available at <https://beta.developer.spotify.com/documentation/web-api/quick-start/> (emphasis added)

## CLAIM 1

validating the one or more mask sets either before or after manipulating the file format information using the predetermined key.

## SPOTIFY

**“a media delivery module 122 for sending (e.g., streaming) media content to a client device (e.g., client device 110-1) remote from sever system 120-1 in response to the request from client device 110-1, media delivery module 122 including but not limited to:**

**an encoding module 318 for encoding media content prior to sending (e.g., streaming) the media content to client device 110-1 or to other content sources for storage;**

**an encryption module 320 for encrypting one or more portions of media content prior to sending (e.g., streaming) the media content to client device 110-1.**

**a compression module 322 for compressing media content prior to sending (e.g., streaming) the media content to client device 110-1;**

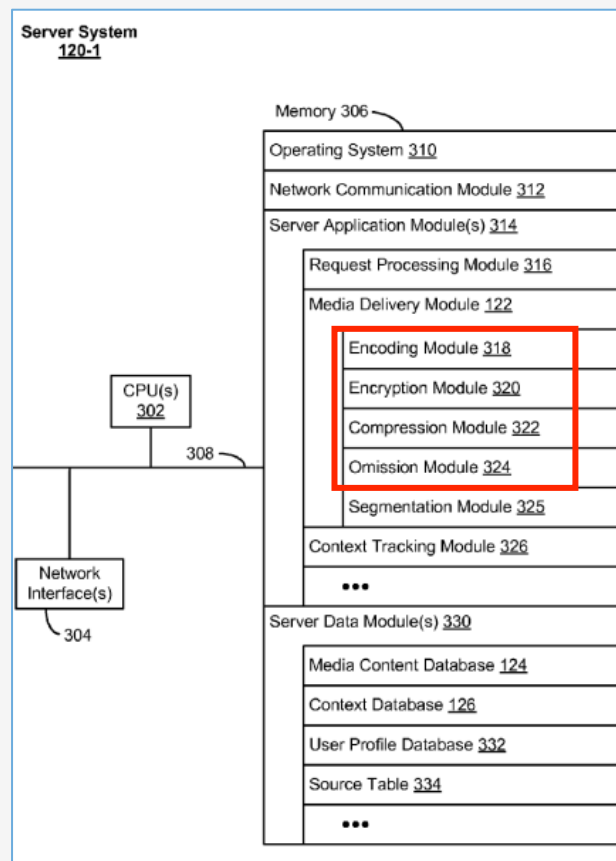
**an omission module 324 for omitting information from media content prior sending (e.g., streaming) the media content to client device 110-1 (e.g., by determining redundant information that can be omitted when compressing the media content while maintaining a quality of the media content above a predefined quality level); and”**

See Exhibit 30, at Col. 11:08-32, “U.S. Patent No. 9,529,888 - System and method for efficiently providing media and associated metadata.” Patent assigned to Spotify (Date Accessed 05/10/2018), available at <https://patentimages.storage.googleapis.com/88/d4/11/6b1fc7c1e09e6b/US9529888.pdf> (emphasis added)

## CLAIM 1

validating the one or more mask sets either before or after manipulating the file format information using the predetermined key.

## SPOTIFY



See Exhibit 30, at Col. 11:08-32 & Fig. 3, "U.S. Patent No. 9,529,888 - System and method for efficiently providing media and associated metadata." Patent assigned to Spotify (Date Accessed 05/10/2018), available at <https://patentimages.storage.googleapis.com/88/d4/11/6b1fc7c1e09e6b/US9529888.pdf> (emphasis added)

## CLAIM 1

validating the one or more mask sets either before or after manipulating the file format information using the predetermined key.

## SPOTIFY

### “How does EME Work?”

1. A web application attempts to play audio or video that has one or more encrypted streams.
2. The browser recognizes that the media is encrypted (see box below for how that happens) and fires an encrypted event with metadata (initData) obtained from the media about the encryption.
3. The application handles the encrypted event:
  - a. If no MediaKeys object has been associated with the media element, first select an available Key System by using navigator.requestMediaKeySystemAccess() to check what Key Systems are available, then create a MediaKeys object for an available Key System via a MediaKeySystemAccess object. Note that initialization of the MediaKeys object should happen before the first encrypted event. Getting a license server URL is done by the app independently of selecting an available key system. A MediaKeys object represents all the keys available to decrypt the media for an audio or video element. It represents a CDM instance and provides access to the CDM, specifically for creating key sessions, which are used to obtain keys from a license server.
  - b. Once the MediaKeys object has been created, assign it to the media element: setMediaKeys() associates the MediaKeys object with an HTMLMediaElement, so that its keys can be used during playback, i.e. during decoding.
4. The app creates a MediaKeySession by calling createSession() on the MediaKeys. This creates a MediaKeySession, which represents the lifetime of a license and its key(s).
5. The app generates a license request by passing the media data obtained in the encrypted handler to the CDM by calling generateRequest() on the MediaKeySession.
6. The CDM fires a message event: a request to acquire a key from a license server.
7. The MediaKeySession object receives the message event and the application sends a message to the license server (via XHR, for example).
8. The application receives a response from the license server and passes the data to the CDM using the update() method of the MediaKeySession.
9. The CDM decrypts the media using the keys in the license. A valid key may be used, from any session within the MediaKeys associated with the media element. The CDM will access the key and policy, indexed by Key ID.

Media playback resumes.

How does the browser know that media is encrypted?

This information is in the metadata of the media container file, which will be in a format such as ISO BMFF or WebM. For ISO BMFF this means header metadata, called the protection scheme information box. WebM uses the Matroska ContentEncryption element, with some WebM-specific additions. Guidelines are provided for each container in an EME-specific registry.

Note that there may be multiple messages between the CDM and the license server, and all communication in this process is opaque to the browser and application: messages are only understood by the CDM and license server, although the app layer can see what type of message (https://w3c.github.io/encrypted-media/#idl-def-MediaKeyMessageType) the CDM is sending. The license request contains proof of the CDM's validity (and trust relationship) as well as a key to use when encrypting the content key(s) in the resulting license.

See Exhibit 8, p. 1, “What is EME \_ \_ Web Fundamentals \_ \_ Google Developers” forum page (Date Accessed 05/02/2018), available at <https://developers.google.com/web/fundamentals/media/eme> (emphasis added)

## CLAIM 1

validating the one or more mask sets either before or after manipulating the file format information using the predetermined key.

## SPOTIFY

*Upon information and belief, Spotify works similarly utilizing the CDM module as explained here.*

### **“Getting a key from a license server**

In typical commercial use, **content will be encrypted and encoded using a packaging service or tool. Once the encrypted media is made available online, a web client can obtain a key (contained within a license) from a license server and use the key to enable decryption and playback of the content.**

The following code (adapted from the spec examples) shows **how an application can select an appropriate key system and obtain a key from a license server.”**

```
var video = document.querySelector('video');

var config = [{initDataTypes: ['webm'],
  videoCapabilities: [{contentType: 'video/webm; codecs="vp09.00.10.08"' }]}];

if (!video.mediaKeys) {
  navigator.requestMediaKeySystemAccess('org.w3.clearkey',
    config).then(
      function(keySystemAccess) {
        var promise = keySystemAccess.createMediaKeys();
        promise.catch(
          console.error.bind(console, 'Unable to create MediaKeys')
        );
        promise.then(
          function(createdMediaKeys) {
            return video.setMediaKeys(createdMediaKeys);
          }
        ).catch(
          console.error.bind(console, 'Unable to set MediaKeys')
        );
        promise.then(
          function(createdMediaKeys) {
            var initData = new Uint8Array([...]);
            var keySession = createdMediaKeys.createSession();
            keySession.addEventListener('message', handleMessage,
              false);
            return keySession.generateRequest('webm', initData);
          }
        ).catch(
          console.error.bind(console,
            'Unable to create or initialize key session')
        );
      }
    );
}

function handleMessage(event) {
  var keySession = event.target;
  var license = new Uint8Array([...]);
  keySession.update(license).catch(
    console.error.bind(console, 'update() failed')
  );
}
```

See Exhibit 8, p. 1, “What is EME \_ \_ Web Fundamentals \_ \_ Google Developers” forum page (Date Accessed 05/02/2018), available at <https://developers.google.com/web/fundamentals/media/eme> (emphasis added)

## CLAIM 1

validating the one or more mask sets either before or after manipulating the file format information using the predetermined key.

## SPOTIFY

“Say **Felix is a movie streaming service that serves up movies over the browser and uses EMEs to protect the content it delivers.** Suppose that user wants to play some encrypted content from Felix. Here’s what happens:

**The browser will load up this content, realize it is encrypted and then trigger some Javascript code.**

This Javascript will take the encrypted content and pass it along to something called the **Content Decryption Module (CDM)**—more on this later.

**The CDM will then trigger a request that will be relayed by Felix to the license server.** This is a Felix-controlled server that determines whether or not a particular user is allowed to play a particular piece of content. **If the license server thinks our user can watch the content, it will send back a decryption key for the content to the CDM.**

**The CDM will then decrypt the content and it will begin playing for the user.”**

The most important **chunk of this whole process is the CDM.** By its nature, the **CDM has to be a piece of code that’s trusted by Felix.** If it isn’t, Felix can’t be sure that it will do what it wants it to. This means that the **CDM has to come with the browser.** There are a couple of different providers for CDMs, including **Google’s Widevine** and Adobe Primetime. Both of these provide the **encryption support necessary for content providers such as Felix to enforce DRM.**

See Exhibit 6, p. 2 & 3, “The most controversial HTML5 extension – LogRocket” blog post (Date Accessed 05/01/2018), available at <https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291> (emphasis added)

*Thus, this claim element is infringed literally, or in the alternative under the doctrine of equivalents.*